Dynamic Energy, Performance, and Accuracy Optimization and Management Using Automatically Generated Constraints for Separable 2D FIR Filtering for Digital Video Processing

DANIEL LLAMOCCA, Oakland University
MARIOS PATTICHIS, University of New Mexico

There is strong interest in the development of dynamically reconfigurable systems that can meet real-time constraints on energy, performance, and accuracy. The generation of real-time constraints will significantly expand the applicability of dynamically reconfigurable systems to new domains, such as digital video processing.

We develop a dynamically reconfigurable 2D FIR filtering system that can meet real-time constraints in energy, performance, and accuracy (EPA). The real-time constraints are automatically generated based on user input, image types associated with video communications, and video content. We first generate a set of Pareto-optimal realizations, described by their EPA values and associated 2D FIR hardware description bitstreams. Dynamic management is then achieved by selecting Pareto-optimal realizations that meet the automatically generated time-varying EPA constraints.

We validate our approach using three different 2D Gaussian filters. Filter realizations are evaluated in terms of the required energy per frame, accuracy of the resulting image, and performance in frames per second. We demonstrate dynamic EPA management by applying a Difference of Gaussians (DOG) filter to standard video sequences. For video frame sizes that are equal to or larger than the VGA resolution, compared to a static implementation, our dynamic system provides significant reduction in the total energy consumption (>30%).

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Optimization*; I.3.1 [**Computer Graphics**]: Hardware Architecture; I.4.3 [**Image Processing and Computer Vision**]: Enhancement—*Filtering*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Dynamic partial reconfiguration, FPGA, 2D separable FIR filtering

ACM Reference Format:
Daniel Llamocca and Marios Pattichis. 2014. Dynamic energy, performance, and accuracy optimization and management using automatically generated constraints for separable 2D FIR filtering for digital video processing. ACM Trans. Reconfig. Technol. Syst. 7, 4, Article 4 (December 2014), 30 pages.
DOI: http://dx.doi.org/10.1145/2629623

1. INTRODUCTION

Digital video architectures need to meet specific, real-time constraints of performance and accuracy. For digital video processing on mobile devices, it is also important to reduce the required power and energy. For each architecture, there are definite tradeoffs among energy, performance, and accuracy (EPA). Yet, it is important to recognize that

Authors' addresses: D. Llamocca, Electrical and Computer Engineering Department, Oakland University, Rochester, MI, 48309; email: llamocca@oakland.edu; M. Pattichis, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, 87131; email: pattichis@ece.unm.edu.

DOI: http://dx.doi.org/10.1145/2629623

EPA requirements can change. For example, for a mobile device that is running out of battery, it is important to switch to a low-power mode. On the other hand, when there is a heightened interest in specific sections of a video sequence, we are more interested in achieving performance and accuracy at the expense of power consumption. More generally, there is interest in dynamically reconfigurable video processing architectures that can meet real-time EPA constraints, where the constraints are automatically generated by the system.

Our development of a dynamically reconfigurable architecture system is based on dynamic partial reconfiguration (DPR) technology that allows management of hardware resources in real time. More specifically, we are interested in the development of a dynamically reconfigurable 2D FIR filtering system for digital video processing applications. Our focus on 2D FIR filtering comes from the large number of applications that can benefit, and also from the need to provide meaningful optimization over a large number of hardware implementations. The list of potential applications includes image and video denoising, linear image and video enhancement, image restoration, edge detection, face recognition, and so forth [Bovik 2009a, 2009b]. Depending on the application, we can have very different EPA requirements. Furthermore, we can have real-time constraints that are imposed during the execution of a particular application.

These real-time video constraints are generated automatically based on three approaches: (1) user-driven optimization, (2) video-communication-based constraints, and (3) video-content-adaptive constraints. In the first case, the user supplies constraint parameters that are used to set up a constrained optimization problem. For example, in the minimum energy consumption mode, the user can provide minimum acceptable levels of accuracy and performance. For video communications, we use the image type in the Group of Pictures (GOP) to set the constraints [Bovik 2009b]. Here, we note the required use of different image types within a GOP in all current and future video compression standards. Within the GOP, video frames are assigned into different types [Bovik 2009b]. Thus, since I-frames are used for predictive encoding of all other frame types, we require the highest accuracy for this frame type. Similarly, we require less, but still high, accuracy for P-frames. B-frames are predictively encoded from all other frame types, thus requiring the least amount of accuracy. For video-content-adaptive constraints, we reserve the highest accuracy for visualizing scene changes while operating in a low-energy mode during routine video viewing. To detect scene changes, an effective approach is proposed based on relative change in the sum of absolute values of the Difference of Gaussians (DOG) filter outputs for each frame. The motivation for the content-adaptive approach relies on the fact that human observers are sensitive to scene changes and will likely want to pay more attention to the video during scene changes. Here, also note that the DOG filter is believed to be used by the human visual system to extract image information [Bovik 2009a].

Dynamic reconfiguration management is based on using DPR to implement precomputed realizations. We are only interested in implementing realizations that are Pareto optimal among all possible hardware realizations [Llamocca and Pattichis 2013] (see Figure 1). Figure 1(b) shows how we select a Pareto-optimal realization that meets or exceeds real-time constraints. More specifically, Pareto-optimal realizations will simultaneously minimize energy and maximize accuracy and performance. When several realizations meet the constraints, we will pick the one that minimizes energy consumption. Dynamic EPA management is achieved by switching among Pareto-optimal realizations based on real-time constraints, as in Figure 1(c).

Dynamic EPA management has been briefly discussed in some earlier work (e.g., [Burleson et al. 2001; Chamberlain et al. 2003]), where it was suggested that one of these three system properties could be potentially modified via DPR. As the design flow

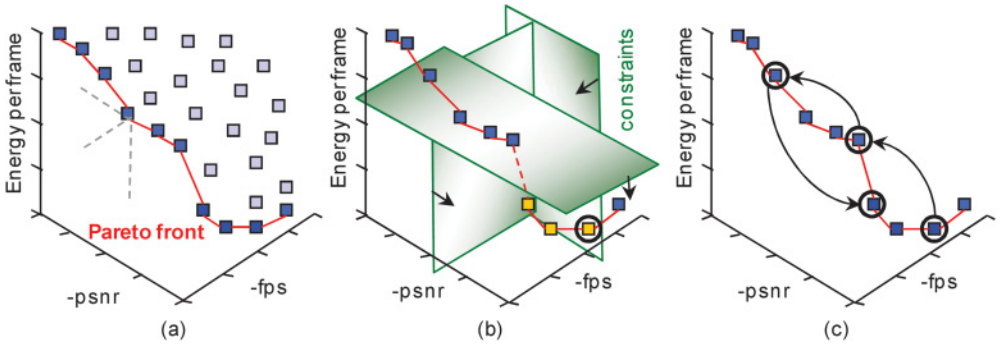

Fig. 1. Multiobjective optimization of the EPA space: (a) 3-D Pareto front. (b) Three constraints applied to the 3D Pareto front. Selected point (min. energy) is circled. (c) Dynamic EPA management: moving along the Pareto front based on real-time constraints.

for DPR matured, more work regarding this has appeared (e.g., [Noguera and Kennedy 2007; Huang and Lee 2009; Vera et al. 2011]). Power, performance, and accuracy were variables commonly changed using DPR by trading off one for the other. A dynamic arithmetic example for controlling precision in real time was presented in Vera et al. [2011]. A thorough investigation of the EPA space for a single-pixel processor was presented in Llamocca and Pattichis [2013].

For efficient hardware realizations, we focus on 2D FIR separable filtering that allows implementations by means of two 1D FIR filters. This separability property allows us to consider a DPR approach that keeps only one filter (row or column) at a time. Furthermore, note that nonseparable 2D filters can be expressed as a sum of separable kernels through the use of Singular Value Decomposition [Andrews 1999]. This technique can be either exact (no error introduced) or inexact (certain approximation error is allowed). Thus, without loss of generality, we focus our attention on separable 2D filters.

The main contributions of the current article include (1) a methodology for automatic constraint generation and satisfaction based on user inputs, image type associated with video communications, and changes in video content; (2) a new set of 2D FIR filters that include isotropic and anisotropic Gaussian filters and DOG with support for both symmetric and nonsymmetric filters; (3) a mathematical formulation of the multiobjective design space, the computational time required to get the Pareto front, and a partial search to reduce the overhead associated with the Pareto front computation; (4) a comparison of our dynamic system with static implementations based on the reconfiguration rate and different video frame sizes; (5) a discussion of memory requirements (Table VII); and (6) an integrated DPR management system that demonstrates the application of these new 2D FIR filters and automatically generated constraints to standard video sequences.

In terms of the 2D FIR filters considered for the first time in this article, we note that the isotropic Gaussian filter is routinely used for low-pass filtering. Similarly, the Difference of Gaussians is an integral part of edge detection and SIFT (an emerging method with wide applications in computer bision and image processing [Lowe 1999; Lowe 2004]).

More importantly, we note that the current article's focus on automatic constraint generation allows automatic hardware adaptation that can be used in a wide variety of applications. For example, our use of MPEG image types can be extended to cover current and emerging video compression standards. Also, video content adaptation is

used to demonstrate real-time switching into low-accuracy (associated with low power), medium-accuracy, and high-accuracy modes. This concept is significant in the sense that it demonstrates an application of the approach in video image analysis.

Our description of the design space, memory requirements, comparison to static approaches, and the presentation of the integrated system will help digital designers assess the complexity and wider applicability of the approach. We expect to see wider applications of dynamically reconfigurable systems that use automatically generated constraints.

Since our 2D FIR filter is implemented via dynamic reconfiguration of 1D FIR filters, our approach depends on the use of an efficient DPR controller. Refer to Huang and Lee [2009], Claus et al. [2008], Liu et al. [2009], and Hoffman and Pattichis [2011] for research related to efficient DPR controller development.

The rest of the article is organized as follows: Section 2 presents background and related work. Section 3 details the dynamic video filter implementation. Section 4 presents the optimization and dynamic management framework for 2D filters. Section 5 presents the experimental setup. Section 6 presents the results. Section 7 lists the conclusions.

2. BACKGROUND AND RELATED WORK

Most image processing implementations on FPGAs focus on static architectures that cannot be reconfigured at runtime. Examples include brain MRI tissue classification [Koo et al. 2009], face detection systems [Nguyen et al. 2006], reconfigurable systems for real-time vision [Komuro et al. 2010] and ultrasonic imaging [Oruklu and Saniie 2009], 2D discrete wavelet transform [Liu and Lai 2004], and binary morphology architectures [Hedberg et al. 2009].

We also provide a summary of 2D FIR filter implementations. Multiply-and-add approaches for 2D separable filters are summarized in Neoh and Hazanchuk [2004] and Turney [2007]. A design methodology that decomposes a 2D filter into 2D separable and nonseparable filters and efficiently allocates the heterogeneous resources (embedded multipliers, LUTs, FFs) on an FPGA is presented in Bouganis et al. [2009]. We provide a side-by-side comparison of 2D FIR implementations from these prior methods in Section 6.1.4. In the results section, we also compare with implementations that use embedded multipliers and demonstrate the effectiveness of our approach. Essentially, though, this article differs from prior work in the extensive use of multiobjective optimization to find the optimal architecture for the application.

Among image processing implementations that make use of DPR, we note a design that dynamically reconfigures among Discrete Cosine Transform (DCT) modules of various sizes [Huang and Lee 2009], a dynamic systolic array accelerator for Kalman and wavelet filters [Sudarsanam et al. 2010], a fingerprint image processing hardware whose stages (e.g., segmentation, normalization, smoothing, etc.) are time multiplexed via DPR [Fons et al. 2010], a system that can reconfigure among single-pixel operations [Llamocca and Pattichis 2013], a 3D Haar Wavelet Transform (HWT) implemented by dynamically reconfiguring a 1D HWT core three times [Ahmad and Amira 2009], and a JPEG2000 decoder that allows the blocks to be dynamically swapped [Bouchoux et al. 2004]. None of these dynamic reconfiguration approaches include the use of multiobjective optimization, automatic constraint generation, and satisfaction.

As for 2D filter implementations using DPR, we have a 2D filterbank based on the coefficient-only reconfiguration of a single 1D FIR filter [Llamocca and Pattichis 2010], a system that switches between a median and an averaging filter on a custom-built FPGA device [Yang et al. 2010], a system that switches between a mean and a median filter [Bhandari et al. 2009], and a system that dynamically reconfigures the coefficients of a 3×3 2D FIR filter [Raikovich and Feher 2010]. By dynamically

reconfiguring between the row and column filter, we implemented a 2D FIR filter in Llamocca et al. [2011] and a 2D complex FIR filter in Llamocca et al. [2012]. Clearly, though, none of these previous approaches include the use of automatic constraint generation and satisfaction that is covered in the current article.

There has also been related work on dynamic EPA management. In Bondalapati and Prasanna [1999], runtime precision management for loop computations was analyzed. In Burleson et al. [2001], the use of reconfiguration is proposed to take advantage of perceptual tolerance in order to dynamically manage power consumption. In Chamberlain et al. [2003], the impact of numerical precision on power consumption is studied for audio processing applications. The work in Hong et al. [2007] presented a static architecture for particle filters that, by tuning buffer parameters and interconnection switches, allowed runtime modification of the number of particles (trading off accuracy and performance) and the degree of parallelism (trading off power and performance). In Vera et al. [2011], the authors explored the use of DPR to dynamically adjust power, performance, and precision on a dynamic arithmetic hardware. In Huang and Lee [2009], the authors showed how they can dynamically adapt DCTs of different sizes based on power, performance, and accuracy considerations. In earlier work in Llamocca et al. [2011], we compared the energy-accuracy space of a 2D FIR filter for FPGA with both DPR and GPU implementations. In Llamocca et al. [2012], we carried out EPA space optimization for complex 2D FIR filters. However, earlier research did not consider automatic constraint generation based on the data types, video content, or user inputs as addressed in the current article.

In the current article, we perform runtime management of the EPA space for 2D filters using a multiobjective optimization approach and basing the decisions on automatically generated real-time constraints. We evaluate different realizations based on their EPA characteristics. Here, each realization comes with its own EPA values. We are only interested in Pareto-optimal realizations [Llamocca and Pattichis 2013]; that is, we select realizations that cannot be improved upon without sacrifice in at least one of the EPA characteristics (see Figure 1(a)). As discussed earlier, the framework allows us to meet automatically generated real-time constraints (see Figure 1(b)).

The reconfiguration time overhead is a limiting factor in the use of DPR. Techniques to reduce this overhead include improving the access speed to the configuration memory [Claus et al. 2008], reducing the size of the reconfigurable area [Hori et al. 2008], efficient runtime task scheduling [Resano et al. 2003], and compressing the bitstreams while they are moved through the slow parts of the system [Huang and Lee 2009]. For the current application, we note that the benefits of dynamic reconfiguration are expected to increase as the sizes of the images increase. This is because image sizes grow and the reconfigured filter is held constant when processing the entire image. Thus, an image size increase by a factor of 2 will approximately require four times more processing time while the reconfiguration overhead is held constant.

Our article seeks to extend prior research in the area of 2D filtering by using the 2D separable FIR filter implementation with DPR presented in Llamocca et al. [2011] and studying its EPA space. Furthermore, we use a multiobjective framework to derive a set of optimal filter realizations over which we can dynamically reconfigure to meet EPA constraints.

3. A VIDEO PROCESSING SYSTEM FOR DYNAMICALLY RECONFIGURABLE 2D FIR FILTERING

In this section, we describe the dynamic 2D separable FIR filter implementation that is based on efficient distributed arithmetic implementations of 1D FIR filters. For EPA optimization in Section 4, we will consider hardware realizations as functions of the number of coefficients, the coefficient bitwidth, and the output bitwidth.

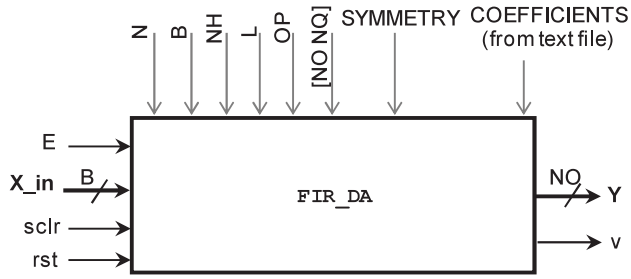

Fig. 2. FIR filter intellectual property (IP) core. Here, N denotes the number of coefficients, NH the coefficients' bitwidth, B the input bitwidth, L the LUT input size (FPGA device dependent), *[NO NQ]* the output fixed-point format, and OP the output truncation scheme.

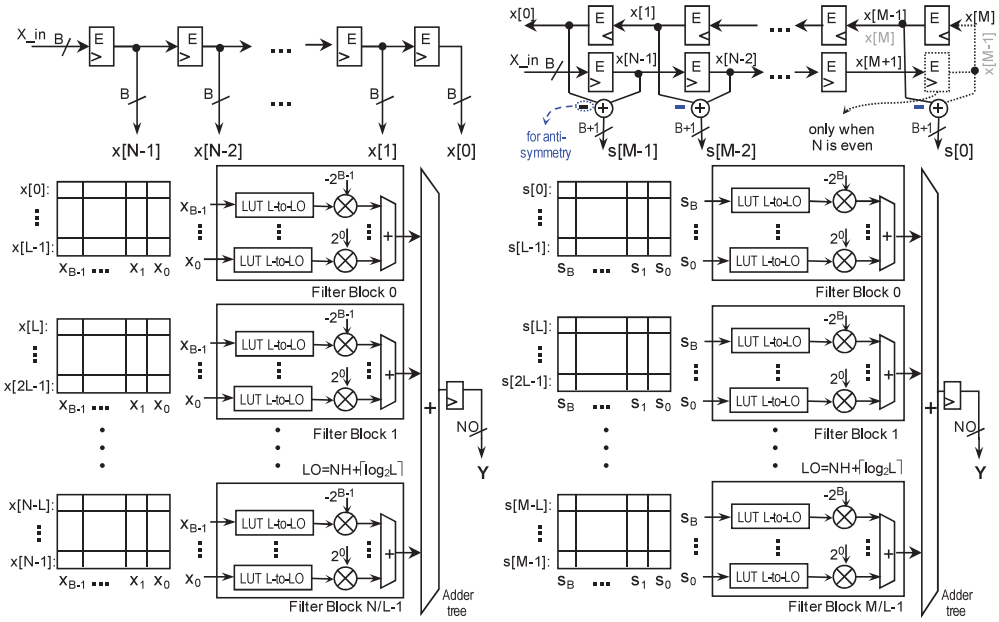

Fig. 3. Distributed arithmetic FIR filter implementations. Nonsymmetric filter (left), symmetric/antisymmetric filter (right). B and NH are independent parameters. Refer to Llamocca et al. [2010] for filter block implementation details. The L-input, LO-output LUT (LUT L-to-LO) is a 2^L-word LUT.

3.1. Implementation and Parameterization for Symmetric, Antisymmetric, and Nonsymmetric 1D FIR Filters

Figure 2 depicts the 1D FIR filter core with its I/O and parameters. More detailed descriptions of the FIR core can be found in Llamocca et al. [2010]. Here, we describe the new (real) FIR core (see Llamocca et al. [2012]), where we allow for the number of input bits (B) to be independent of the coefficients bitwidth (NH). Also, besides symmetric and nonsymmetric filters, antisymmetric filters are supported in the new core (see Figure 3). Table I provides a description of the parameters. This parameterized core can be used to implement either the row or the column filter.

We next consider the largest possible output format based on the input and coefficient formats. We first let NO represent the number of output bits with NQ fractional bits. The output format is then expressed as [NO NQ]. Then, we let the fixed-point input format be [B B – 1] and the coefficients' format be [NH NH – 1], with normalized

Table I. Description of Parameters: 1D FIR Filter Intellectual Property (IP) Core

Parameter	Description
B	Input bitwidth
NH	Number of bits per coefficient
L	LUT input size [Llamocca et al. 2010]
SYMMETRY	Filter symmetry: nonsymmetric (NO), symmetric (YES), antisymmetric (AYES)
COEFFICIENTS	Text file containing the coefficients
[NO NQ]	Output format: NO bits with NQ fractional bits
OP	Output truncation scheme [Llamocca et al. 2010]

inputs/coefficients to be in the interval of [–1,1). The required largest output format is then given by

$$[NH + B - 1 + \lceil \log_2(N+1) \rceil, NH + B - 2]. \quad (1)$$

In cases where we need smaller output formats, overflow is avoided by performing LSB truncation and saturation (controlled by parameter OP).

The FIR filter I/O latency (register levels between input and output) results in $REG\_LEVELS = \lceil \log_2 sizeI \rceil + \lceil \log_2 (M/L) \rceil + 2$ cycles, where (1) $M = \lceil N/2 \rceil$ and $sizeI = B + 1$ for symmetric/antisymmetric filters, and (2) $M = N$ and $sizeI = B$ for nonsymmetric filters.

3.2. A Dynamically Reconfigurable System for 1D FIR Filtering

The constant-coefficients filter is turned into an efficient and flexible FIR filter via DPR, as described in Llamocca et al. [2010]. Two dynamic realizations are considered: coefficient-only and full-filter reconfiguration. We focus on the full-filter reconfiguration case, where the entire filter is included in the Partial Reconfiguration Region (PRR). This lets us generate many realizations for exploring the EPA space since we can modify all the parameters.

Figure 4(a) shows an embedded system that allows for full-filter reconfiguration. The logic outside the PRR is called the static region. The FIR filter processor and the processor interact via the 32-bit Fast Simplex Link (FSL) bus. The bitstreams (filter realization data) and input data are initially stored in the Compact Flash (CF) card. At runtime, the main memory holds input and output data and the bitstreams. The Ethernet link allows for data communication, throughput measurements, and system status reports. To perform DPR, the bitstreams in memory are streamed to the ICAP port [Vera et al. 2011]. We included the FSL interface in the PRR in order to allow changes to the FIR filter I/O bitwidth during DPR.

An FIR filter convolution generates NX + N – 1 values, where N is the number of coefficients and NX the input stream length. The FSL interface offers four filter output choices: (1) basic: first NX output samples, (2) centered: NX samples in the range $\lceil N/2 \rceil + 1 : NX + \lceil N/2 \rceil$, (3) full: all of the NX + N – 1 samples, and (4) streaming: $NX = \infty$, infinite output samples. Thus, we can dynamically control two parameters: input stream length (NX) and the filter output choice ($mode$).

For proper DPR operation, we include a DPR control block that addresses two issues that arise when the FSL interface is inside the PRR: (1) the PRR outputs toggle during DPR, and they are disabled to avoid erratic FSL behavior, and (2) unlike in full reconfiguration, the PRR flip-flops are not automatically reset after partial reconfiguration [Xilinx 2010a]. The DPR control block resets the PRR flip-flops after each partial reconfiguration.

Figure 5 shows the dynamic FIR filter as an FSL peripheral. The FSL bus uses two FIFOs (*FIFOw* and *FIFOr*) to communicate with the FIR filter core. For the 32-bit FSL bus, three I/O interface cases are supported: B = NO = 8, B = NO = 16, and B = 8,

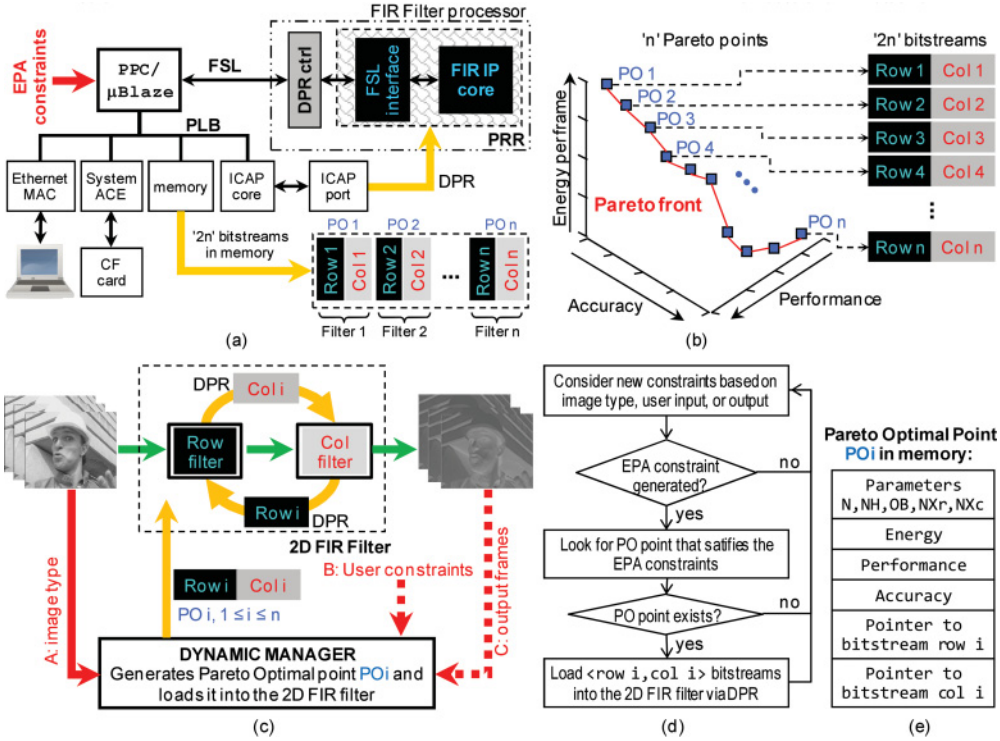

Fig. 4. Dynamic EPA management for 2D FIR filters: (a) Embedded system that allows for dynamic EPA management via DPR. The memory holds the "2n" unique bitstreams needed for the Pareto front. (b) Pareto front with "n" points. (c) Conceptual implementation of dynamic EPA management. The dynamic manager can modify the 2D FIR filter by loading new bitstreams. (d) Dynamic manager operation. It is assumed that the rules to generate EPA constraints based on user inputs and input and output frames have been previously set. (e) Pareto-optimal point in memory (parameters listed in Table II).

NO = 16. Note that the PRR block is depicted here along with its I/Os: 35 input bits and the 34 output bits.

3.3. Dynamic Reconfiguration for Effective Implementation of 2D Separable FIR Filtering

In the context of the embedded system of Figure 4(a), a 2D separable FIR filter (represented by two bitstreams: row and column filter) is implemented by cyclically swapping the row filter with the column filter via DPR [Llamocca et al. 2011]. This approach only requires resources for a single 1D FIR filter at a time and can yield significant savings over static implementations of 2D filters. The following considerations need to be addressed:

—The 2D filtering process usually requires the output image to be of equal size to the input image. Thus, the FSL interface of the FIR filter IP selects the centered output mode.
—The column filter and row filter differ not only in the coefficients, but also in other parameters (I/O format, number of coefficients, etc.). This requires a full filter reconfiguration.
—The length of the input signal in the row filtering case is different than in the column case (unless the image is square). This means that we have to change NX to match the input size.

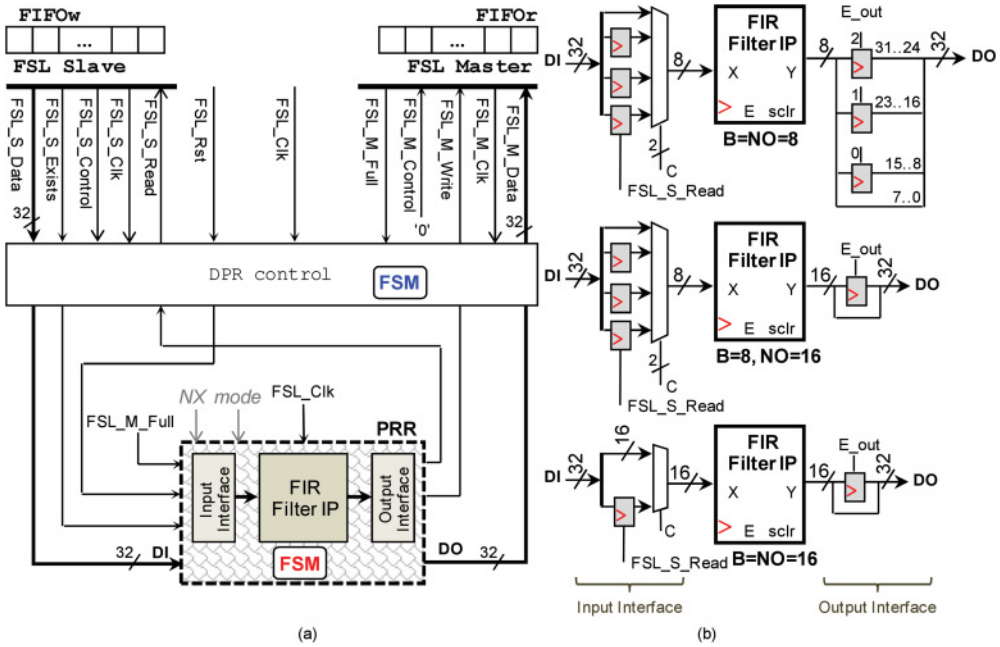

Fig. 5. (a) Dynamic FIR filter as an FSL peripheral. The PRR, which includes the filter core and the FSL I/O interface, has two parameters: input stream length (*NX*) and *mode* (basic, centered, full, streaming). The *mode* determines the state machine. (b) Three I/O interface supported cases: B = NO = 8, B = NO = 16, and B = 8, NO = 16.

—The row filter processes and stores the result in a sequential row-by-row fashion, but the column filter does so in a sequential column-by-column fashion. As a result, the row-filtered output images need to be transposed prior to column filtering.

The cyclic swapping between the row and column filter usually implies a reconfiguration rate of two reconfigurations per frame. Besides the input and output frames, we require memory for one intermediate row-filtered frame.

Alternatively, the row filter can generate and store K row-filtered frames, which need to be transposed. Then, the column filter will generate and store K output frames. Under this scenario, we have a reconfiguration rate of two reconfigurations per K frames. Besides the input and output frames, we would require memory for K intermediate row-filtered frames.

Thus, the reconfiguration rate of the 2D separable FIR filter results in two reconfigurations per K frames. We will see later that the case K = 1 is the less problematic. Also, for nonseparable 2D filters, we note that efficient implementations are based on their approximation using a sum of 2D separable filters [Andrews 1999].

3.4. Extension to Filterbank Implementations

To implement filterbanks, we would have to extend the cyclic approach by sequentially applying DPR to move along each 2D filter's row and column bitstreams. There would not be an extra reconfiguration penalty since we are always reconfiguring twice per K frames (usually K = 1 is preferred). The execution time grows linearly with the number of 2D separable filters. We refer to Llamocca and Pattichis [2010] for a filterbank example (coefficient-only reconfiguration).

4. AN INTEGRATED OPTIMIZATION FRAMEWORK FOR VIDEO FILTERING

This section describes a framework for extracting an optimal set of 2D filter realizations from the EPA space and selecting optimal realizations that meet dynamic EPA constraints. We begin with a description of the Pareto space. As we detail later, we use a partial search to explore the Pareto space and provide a summary of the parameter search in Table II.

To describe the size of the Pareto space, we adopt the same notation as found in Dandekar et al. [2008]. Here, assume that we have P design parameters p_i, $I = 1, 2, \ldots, P$, with each parameter taking values from a corresponding set of valid values v_i. The design space S consists of all P-tuples generated by the Cartesian product of the sets v_i. Thus, the size $|S|$ of the design space is given by the product of the sizes of each set v_i: $|S| = |v_1| \times |v_2| \times \cdots \times |v_P|$, where $|v_i|$ is the number of elements in each set v_i. We also have m objective functions: $f_1, f_2, \ldots f_m$. In the case of the EPA space, $m = 3$. Each function associates a real value for every point (P-tuple) in the design space.

The creation of the design space S involves the evaluation of the m objective functions for each point in the design space, which is assumed to take t_{EVAL} time. Then, the generation of the design space takes $|S| \times t_{EVAL}$. The straightforward generation of the Pareto-optimal front takes $|S|^2 \times m \times t_{CMP}$, where t_{CMP} is the time it takes for a comparison of one operating point with another. We will see that t_{EVAL} can take many minutes, so it is extremely important to reduce $|S|$ as much as possible. On the other hand, t_{CMP} is in the order of μs, so extracting the Pareto front is not a problem if the design space is available.

Dandekar et al. [2008] recommended four methods for obtaining the Pareto front once the design space has been formulated: exhaustive search, partial search, random search, and the use of evolutionary techniques. Exhaustive search is guaranteed to get the Pareto front, but it can require significant computational overhead. Instead of exhaustive search over the entire space, we will focus on searching over a realistic set of values. Furthermore, we can further reduce the design space by considering the use of alternative values from a subset of the realistic parameter values. Thus, our approach is to perform a complete search on a reduced design space. This method, called partial search (or sampling of the design space), is carried out offline. Furthermore, the parameterized VHDL code of the FIR filter IP helps to automate the search.

We next detail (1) the formulation of the set of 2D FIR filter realizations; (2) creation of the EPA space: energy, performance, and accuracy values; (3) generation of Pareto-optimal filters; and (4) the dynamic EPA management that meets EPA constraints. In terms of the cyclic swapping of row and column filters, we choose to work under the scenario with two reconfigurations per frame ($K = 1$) for reasons explained in Section 4.6.

4.1. Formulation of the Set of 2D Filters

The set (or space) of 2D FIR filter realizations, from which the optimal set will be extracted, is generated by varying the FIR filter parameters. We restrict the number of parameters for each 2D filter by keeping N, L, and NH the same for both the row and column filters, while B and [NO NQ] can be different. By varying the input stream length *NX* (usually different for the row and column filters), we can explore different frame sizes. We use the term *EPA space of realizations* to describe the collection of all possible EPA values.

4.2. Performance Calculation

We report the performance of the 2D filtering process from the IP angle, that is, assuming a continuous streaming of the input signal and maximum reconfiguration speed.

The embedded system performance depends on many factors (cache size, processor type, bus usage, etc.) that can easily change. Here, the embedded system is just a generic testbed.

4.2.1. Filter Processing Time. The 2D filter operates on a stream-by-stream basis. After each stream (a single row or column) is processed, the register chain in the FIR filter is cleared, ready for a new stream. Let the lengths of the input streams be defined as NXr and NXc for the row filter and column filter, respectively (input video frame of size NXc rows by NXr columns). The processing time is then given by:

$$\begin{array}{l} t_{rows}\,(sec) = (NXr + \lfloor Nr/2 \rfloor + REG\_LEVELSr) \times NXc \times T_{cycle} \\ t_{cols}\,(sec) = (NXc + \lfloor Nc/2 \rfloor + REG\_LEVELSc) \times NXr \times T_{cycle} \end{array}, \tag{2}$$

where Nr, Nc represent the number of row and column filter coefficients, respectively. $REG\_LEVELSr$, $REG\_LEVELSc$ denote an initial latency (see Section 3.1), and T_{cycle} is the clock period. Here, note that $\lfloor Nr/2 \rfloor$, $\lfloor Nc/2 \rfloor$ cycles are needed to provide centered row/column convolution outputs.

4.2.2. Reconfiguration Time. Based on the PRR bitstream size and the reconfiguration speed, the reconfiguration time results in

$$t_{reconfig}\,(sec) = PRR\ size(bytes)/Rec.speed(bytes\ per\ sec). \tag{3}$$

The maximum reconfiguration speed (400MB/s in Virtex-4) is achieved with a direct link between the ICAP port and the BRAMs (local memory inside the FPGA) that holds the bitstreams [Hori et al. 2008; Liu et al. 2009]. The approach is limited by the available BRAMs. While BRAMs are scarce in Virtex-4 devices, there are far more BRAM resources in the (newer) Virtex-6 devices [Xilinx 2010b], making the approach more practical. We will assume the maximum reconfiguration speed for reporting our performance results.

4.2.3. 2D Filter Performance. Filtering a frame of $NXr \times NXc$ pixels entails row, column filtering, and two reconfigurations. The performance (in frames per second) is given by

$$fps = 1/(t_{rows} + t_{cols} + 2 \times t_{reconfig}). \tag{4}$$

In Equation (4), the transposition time for the row-filtered image is not considered. The transposition operation is a software routine in the embedded platform [Llamocca et al. 2011]. For completeness, we report the transposition time in the embedded results section.

4.3. Energy Measurement Estimation

In this section, we report the energy spent by the 2D FIR filter and the reconfiguration process for processing a single frame. We focus on energy per frame as it provides more information than power since 2D filtering has several stages (row filtering, column filtering, reconfiguration) that draw different amounts of power.

4.3.1. Power Measurements. Power inside the FPGA is drawn by the following power supply rails: (1) internal supply voltage VCCINT with current ICCINT, and (2) auxiliary supply voltage VCCAUX with current ICCAUX. The output supply power is not considered since it is only associated with the power drawn by the external pins.

Power at each supply rail is divided into static and dynamic power. The static power is drawn by the device when it is powered up, configured with user logic, and has no switching activity. It is divided into (1) device static power: drawn by the device when it is powered up and not programmed, and (2) design static power: drawn by the user logic when the device is programmed and with no switching activity. The dynamic power is

the fluctuating power as the design runs; it is generated by the switching user logic and routing [Xilinx 2011].

The device static power depends on the operating conditions and the device size/family. FPGA datasheets provide constant device static currents (at nominal voltage, 25°C) for each supply rail. For the XC4VFX20 Virtex-4 FPGA, the device static currents are given by ICCINTQ = 71mA and ICCAUXQ = 35mA. For comparing among different cases, we will report the IP power as the sum of the dynamic and design static power (ignoring the device static power). The IP core power for the row/column filter then results in

$$P_{row/col} = VCCINT \times (ICCINT - ICCINTQ) + VCCAUX \times (ICCAUX - ICCAUXQ). \tag{5}$$

Direct power measurement (e.g., [Vera et al. 2011]) requires custom-built boards that provide access to the voltage rails. Instead, we estimate power consumption (at 25°C) with the Xilinx Power Analyzer (XPA). The XPA provides an accurate estimate based on simulated switching activity of the place-and-routed circuit and exact utilization statistics [Xilinx 2011]. More generally, power estimation in reconfigurable systems is an area of active research. Becker et al. [2003] report XPA accuracies of 70% to 94% for dynamic power consumption. The accuracy strongly depends on proper modeling of input switching activity and environmental conditions (junction temperature, airflow, etc.).

As for power consumption during partial reconfiguration, no software tool can provide an accurate estimate. Measurement and modeling of reconfiguration power is an active area of research [Bonamy et al. 2012]. In Vera et al. [2011], by direct current measurements, it was found that only ICCAUX was increased (by 25mA) during DPR (Virtex-4 XC4VFX12). Assuming that this dynamic current remains the same within the Virtex-4 device family, we will use the same approach (and current value) for obtaining the reconfiguration power estimate. The DPR power (power drawn by the user logic and the increase due to DPR) is thus estimated using

$$P_{reconfig-row/col} = P_{row/col} + VCCAUX \times ICCAUX(increase). \tag{6}$$

4.3.2. Energy per Frame. The total energy per frame is the sum of the energy spent by the following processes: (1) row filtering, (2) turning a row filter into a column filter via DPR, (3) column filtering, and (4) turning a column filter into a row filter via DPR. Using the power and the processing times of each process, the average energy per frame ("epf," in Joules) is given by

$$epf = P_{rows} \times t_{rows} + P_{cols} \times t_{cols} + \left(P_{reconfig-row} + P_{reconfig-col}\right) \times t_{reconfig}. \tag{7}$$

4.4. Accuracy Measurements

We measure the accuracy of the filtered images using the peak signal-to-noise ratio (PSNR) between the fixed-point filter output from the FPGA and the 2D filter output using double floating-point precision. Here, we note that this is the most time-consuming task of the design space generation, as it requires generating unique partial bitstreams for each 2D FIR filter in the design space and then streaming an input image over each filter, thereby getting the output frames needed to compute the actual accuracy values.

4.5. Generation of Optimal Filter Realizations

Based on energy per frame, performance, and accuracy (EPA) values, we create the EPA space. Here, we note that while performance can be quickly calculated, energy estimation of each hardware configuration is in the order of minutes. Also, accuracy

measurements can take even more time since we need to obtain an output frame for each 2D FIR filter in the design space.

Using the EPA space, we can get the optimal realizations. Recall that a 2D filter realization is defined to be optimal in the multiobjective (Pareto) sense if we cannot improve on its EPA values without deteriorating on at least one of them [Llamocca and Pattichis 2013].

Our goal is to minimize the energy per frame consumption and to maximize performance and accuracy. The set of Pareto-optimal realizations is known as the Pareto front. Figure 1(a) shows the EPA space and the Pareto front. The points are plotted against energy and the negatives of performance and accuracy. The computation of the Pareto-optimal points is straightforward. We are interested in understanding how the FIR core parameters generate Pareto-optimal realizations.

4.6. Dynamic EPA Management Using Real-Time Automatically Generated Video Constraints

Dynamic EPA constraints can be met by selecting solutions along the Pareto front. Figure 1(b) depicts a case with EPA constraints (maximum allowed energy per frame, minimum levels of performance and accuracy). Independent constraints are represented as planes in 3D. The golden points (feasible set) are the realizations that meet those constraints. The minimum energy realization is the one finally selected.

The embedded system of Figure 4(a) allows for dynamic EPA management of a 2D FIR filter via DPR. The system can move dynamically along the Pareto front, as shown in Figure 1(c). Note that switching from one 2D FIR filter to another does not incur any time or energy penalty as we are always reconfiguring twice per frame.

We consider three different methods for generating real-time constraints on the EPA space: (1) constraints based on image type that relate to video communications, (2) user-defined constraints, and (3) constraints based on video content. Figure 4(c) illustrates how these constraints are used in the overall system.

First, for video communications, the input image type can be used to determine appropriate constraints (e.g., see "A" in Figure 4(c)). In this case, we assume that the final output will be encoded for video communications. For example, based on image types associated with the MPEG GOP, we assign hardware realizations as follows: (1) highest accuracy for I-frames, (2) high accuracy for P-frames, and (3) low accuracy for B-frames (see discussion in the introduction).

Second, the user can dynamically set the constraints. Here, we are primarily interested in dynamic constraints on energy. For example, users may want to impose a low-energy constraint to prolong battery life.

Third, we consider the generation of dynamic constraints based on the output video content. The basic idea is to compute more accurate results during scene changes. For this example, we consider the use of a DOG filter. The percentage change of the DOG filter output is then used as a detector of possible scene changes. Thus, in this example, we detect scene changes using

$$\left| \frac{\sum_{i,j} |y_n(i,j)| - \sum_{i,j} |y_{n-1}(i,j)|}{\sum_{i,j} |y_n(i,j)|} \right| > T, \tag{8}$$

where $y_n = I_n * DOG$ denotes the nth output frame, I_n the nth input frame, (i, j) the image pixel index, and T a threshold value. In what follows, we will assume that the DOG filter is a required part of the video image analysis system that we are implementing and its computation does not represent an overhead of the DPR system. It represents a 2D separable FIR filter that we are implementing with the current system. Thus, the only overhead incurred by the dynamically reconfigurable system is due to the threshold calculation of Equation (8). Here, we note that this computation

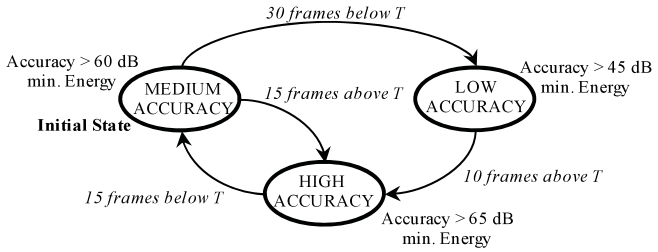

Fig. 6. DPR manager based on the output frames. Transitions are controlled by counting the number of contiguous frames whose sum of absolute values stay above or below a threshold T. The number of video frames for each transition assumes that we have 30 frames per second.

is carried out in software in the PowerPC and its overhead should be considered to be negligible compared to the required computations associated with the DOG filter itself. Here, the use of the DOG filter for detecting scene changes can be justified based on its standard use for edge enhancement/detection that is also motivated by the fact that it provides a model for the function of ganglion cells of the human retina [Bovik 2009a]. To understand how Equation (8) works, note that large scene changes are often associated with the introduction of new image edges, which will certainly induce large differences in the relative DOG filter outputs that are captured in Equation (8). Also, since we are measuring percentage change, we can heuristically set T to a standard normalized value that should work for most videos. As we discuss later, $T = 0.01$, which represents that a 1% change worked well in our scene change application. If greater sensitivity to scene changes is needed, then the users can reduce this value.

The basic DPR manager based on the output frames is shown in Figure 6 with three states: (1) high-accuracy state for scene changes, (2) medium-accuracy state, and (3) low-accuracy/low-energy state for persistent scenes. For transitions between states, we look for persistency by requiring that a number of frames stay above a certain threshold. This helps avoid incorrect transitions while providing some time for the human observers to adjust to the scene. Clearly, though, our approach can be modified and extended for more accurate applications on large-scale databases. The values shown in Figure 6 were heuristically set for videos viewed at 30 frames per second. The number of frames in the transitions shown in Figure 6 will have to be adjusted based on the video frame rate. For example, at twice the frame rate, we will have to double the number of video frames in each transition. Furthermore, to achieve more energy savings, we would significantly increase the number of frames required to remain above T for leaving the low-accuracy state. On the other hand, the accuracy levels (in dB) should remain somewhat invariant to the application.

Figure 4(d) details the dynamic manager operation. Figure 4(e) shows the structure of a Pareto-optimal point in memory. A 2D FIR filter Pareto-optimal realization is represented by its associated parameters, two bitstreams (row and column filters), and its EPA values.

As for the cyclic swapping of row and column filters, we can control the reconfiguration rate by allowing two reconfigurations every K frames (K $>$ 1), where K can be adjusted to achieve different reconfiguration rates. If a constraint is triggered based on the input image type, unless we can group together K frames of the same type (e.g., as in the case of the many B-pictures), we will have to interrupt the row filtering of K frames, complete column filtering on the available row-filtered frames, and then switch to a different filter on the next frame. For constraints based on user input, the same applies if the constraint is triggered during row filtering. If, however, the constraint is triggered during column filtering, we would have to discard some row-filtered frames

Table II. Parameter Combinations (108) for the Set of 2D Filters. The choice of $OB = 8,16$ is based on the fact that the FSL bus width is 32 bits. We fix the LUT input size (L) for a given N.

Frame Size (NXc × NXr)	640×480 (VGA)		352×288 (CIF)		176×144 (QCIF)	
Number of coefficients (N) (LUT input size (L))	8 (4)	12 (6)	16 (4)	20 (5)	24 (6)	32 (4)
Coefficients bitwidth (NH)	10		12		16	
Output bitwidth (OB)	8				16	

and switch to a different filter on the next frame. To trigger a constraint based on output frames, we must wait until we get the K output frames. So, we cannot switch to a different filter in the next frame (in the worst case, we would be off by K – 1 frames).

We also note that in the case that the constraints are only imposed after processing $n \times K$ video frames, where n is an arbitrary nonnegative integer, we will not be discarding or losing frames. For example, in Figure 6, state transitions only occur on multiples of K = 5 video frames. In this case, for K = 1 or 5, we are not losing any frame. However, we do note that both the delay and memory requirements grow linearly as a function of K.

In terms of the worst-case analysis, it is clear that the case for K = 1 will produce the maximum amount of reconfiguration overhead while allowing for the maximum flexibility. For generating the EPA space, we will consider the worst case with the understanding that energy consumption can be reduced and processing performance can be improved by reducing the reconfiguration rate.

5. EXPERIMENTAL SETUP FOR DIGITAL VIDEO PROCESSING

5.1. Generation of the Set of 2D Separable Gaussian and Difference of Gaussians Filters

We test our FIR core platform using Gaussian filters of different spreads (sigmas) and frequency characteristics. The following Gaussian filters are selected: (1) isotropic, low-pass filter with $\sigma = 1.5$; (2) anisotropic, low-pass filter with $\sigma_x = 4, \sigma_y = 2$; and (3) isotropic, band-pass filter based on a DOG, with $\sigma_1 = 2, \sigma_2 = 4$. Note that all of these 2D filters are separable and symmetric.

5.2. Selection of Practical 2D FIR Filter Core Parameters for Different Video Frame Sizes

For the filters, we use symmetric implementations, with an arithmetic mode that uses truncation of the LSB, followed by saturation. In all cases, we consider 8-bit input images (for row filter input bitwidth). We also constrain the output to be in the same range as the input $[-1, +1)$ (NQ = NO – 1) and match the number of output bits of the row and column filters. We define OB as the 2D filter output bitwidth. In the context of the 32-bit FSL peripheral, OB can take two values. OB = 8 requires a row filter with B = NO = 8 and a column filter with B = NO = 8. OB = 16 requires a row filter with B = 8, NO = 16 and a column filter with B = NO = 16. OB = 8,16 requires us to employ the three supported cases of the FSL peripheral (see Figure 5).

For simplicity, we keep N, L, and NH the same for both the row and column filters. Here, for the anisotropic cases, note that accuracy could be improved if we considered separate numbers of coefficients for each dimension (as a function of sigma). For the isotropic cases, the optimal case is to keep the number of coefficients the same as we do here. Table II summarizes the choice of parameter values (that sample the design space) for N, NH, OB, and frame size (NXr × NXc).

5.3. Offline Computation Time for EPA Space Generation and Pareto Front Extraction

We have to create three design spaces, one for each Gaussian filter. Using Table II, we see that the size $|S|$ of the design space for one filter results in $3 \times 6 \times 3 \times 2 = 108$

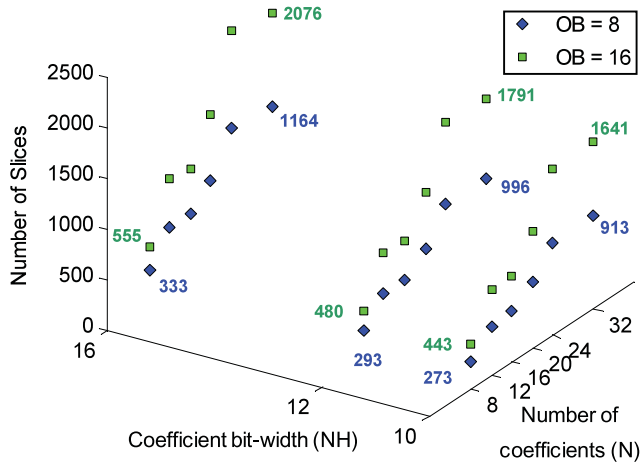

Fig. 7. Hardware resource utilization for the column filters. For all filter types, the resources depend on the number of coefficients (N), the coefficient bitwidth (NH), and the output bits (OB). Hardware usage for the row filters is very close to that of the column filters for OB = 8 (see text for details).

points. On average, we spent $t_{EVAL} = 30$ minutes to get each point (synthesis, place and route, power estimation, partial bitstream generation, and testing); thus, the creation of one design space takes about 54 hours. Partial bitstream generation is the most time-consuming task, but the task can easily be distributed among several computers. So, even for small design spaces, the computational time can quickly become unreasonable; thus, it is imperative to have a reduced design space. The extraction of the Pareto front from one EPA space (once this is available) takes about $1082 \times 3 \times (t_{CMP} = 1\mu s) =$ 35ms.

5.4. Platform Testing Scheme

The 2D FIR filtering system of Figure 4(a) was implemented on the ML405 Xilinx Development Board with a XC4VFX20-11FF672 Virtex-4 FPGA. The selected processor (PowerPC) is clocked at 300MHz, peripherals run at 100MHz, and the memory (128MB DDR-SDRAM) space cached.

6. RESULTS AND ANALYSIS

6.1. Hardware Resource Utilization

6.1.1. FSL FIR IP Utilization. Figure 7 presents the numbers of slices required by the stand-alone column filter core (PRR and the DPR control block) as a function of the number of coefficients, coefficient bitwidth, and output bits. Refer to Table II for parameter combinations. Here, for OB = 8, the row filter (B = NO = 8) resources are very close (difference of two to five slices) to those of the column filter. Interestingly, the row filter with B = 8, NO = 16 incurs negligible extra resources (one to eight slices) compared to the row filter with B = NO = 8. Thus, we can say that the column filter results for OB = 8 in Figure 7 are very similar to the row filter results for either OB = 8 or OB = 16. Furthermore, we directed the ISE synthesizer to avoid optimizing on the LUT contents, thus providing the same resource consumption for the three filter types.

The column filter results of Figure 7 use NX = 480 and *mode* = "centered." The frame size has almost no effect on the resource consumption of the logic within the

Table III. Embedded FIR Filtering System Resource Utilization (Virtex-4 XC4VFX20-11FF672). Largest column filter: N = 32, NH = 16, OB = 16

Module	Slice	(%)	FF	(%)	LUT	%
PRR (column filter)	2,125	25%	3,680	21%	3,812	22%
Static Region	4,973	58%	5,226	31%	5,998	35%
Overall	7,098	83%	8,906	52%	9,810	57%

PRR: larger frame sizes only increase NX, which in turn increases the size of a counter and a comparator in the FSM.

The range of required resources in Figure 7 varies significantly. For example, the use of eight 10-bit coefficients with 8-bit output bits requires the minimum of 273 slices. The use of thirty-two 16-bit coefficients with 16-bit output bits requires the maximum of 2,076 slices. This strong variation in resource consumption will enable an effective generation of the EPA space.

6.1.2. Size of Reconfigurable Area. The PRR size is set to the largest possible filter realization (be it row or column). This largest realization is given by the column filter with N = 32, NH = 16, and OB = 16. The PRR occupies a tightly packed area of $24 \times 94 = 2256$ slices for a bitstream size of 183,754 bytes.

6.1.3. Embedded System Resource Utilization. Table III shows the hardware resource usage of the embedded FIR filtering system of Figure 4(a) that can implement all the cases listed in Table II. The PRR includes a 1D filter and the FSL interface. The largest realization occupies 2,125 slices (94% of the allocated space for the PRR). This is slightly higher than what Figure 7 reports since the results are obtained within the context of the embedded system and the FIR filter peripheral is constrained to the PRR area. For transposition of the row-filtered image, we have 4,152μs, 1,453μs, and 379μs for the VGA, CIF, and QCIF frame sizes, respectively.

For reconfiguration, we used the Xilinx ICAP core, obtaining an average reconfiguration speed of 16.28MB/s (resulting in a reconfiguration time of 11.28ms). Significant improvements can be obtained with the use of custom-built controllers as reported in Claus et al. [2008] and Liu et al. [2009].

6.1.4. Comparisons with Other Systems. Table IV compares our approach with related 2D FIR implementations. For comparison purposes, we picked the symmetric filter with N = 16, NH = 16, and OB = 16 that requires a 120KB bitstream (if this filter were the largest realization).

A fundamental difference between our 2D implementation and the ones reported in Table IV is that we use a distributed arithmetic approach as opposed to multiply-and-add-based methods. Another advantage of our approach is that, at all times, we only allocate resources for one 1D filter. The use of separable filtering by other approaches requires the allocation of resources for both the row and column filters at all times.

A block-wise implementation of a 2D separable FIR filter is given by a Xilinx reference design [Turney 2007] shown in the last column of Table IV. Here, we note that our approach requires more slices (it uses more bits per coefficient) but saves on the use of expensive DSP48 blocks and BRAM resources. The input and output are image blocks whose size is that of the 2D FIR filter kernel. An embedded processor is required to read the image block-wise and pad it (to get the correct output block), making memory addressing complicated and inefficient (as opposed to raster scanning an image). This adds complexity to the embedded peripheral and the software routine.

The filter reported by Neoh and Hazanchuk [2004] uses an ALTERA device that makes use of Adaptive LUTs (ALUTs) that can pack more logic than a Xilinx slice. However, as for the Xilinx case, this implementation uses 16 DSPs. Similarly, our

Table IV. Implementation Comparisons for 2D FIR Filters

	DPR-DA (Ours)	Neoh and Hazanchuk	Yang et al.	Bhandari et al.	Raikovich and Feher	Turney
Filter type	NC×NR	7×7 Gaussian	5×5 mean	5×5 mean	3×3	NC×NR
Separable	Yes	Yes	Yes	Yes	No	Yes
Coefficients	Variable at runtime via DPR	Fixed	Fixed	Fixed	Variable at runtime	Variable at compilation time
Size	Variable at runtime	Fixed	Fixed	Fixed	Fixed	Variable at compilation time
Implementation	DA based	Multiply-and-add	Multiply-and-add	Multiply-and-add	Multiply-and-add	Multiply-and-add
Test case	16×16 symmetric 8-bit input, 16-bit coeffs, 16-bit output	Symmetric filter, 8-bit input	Symmetric filter, 8-bit input	Symmetric filter, 8-bit input	8-bit input	15×15 symmetric 8-bit input, 12-bit coeffs, 12-bit output
Device	Virtex-4	StratixII	Custom-made FPGA	Virtex-4	Virtex-5	Virtex-II Pro
Bitstream size (DPR cases)	120KB	N/A	28KB	242KB	N/A	N/A
Resources	1,098 slices 0 BRAMs 0 DSP48	320 ALUTs 7 M4K 16 DSP	246K logic gates	3,410 slices 5,590 LUTs 5 BRAMs	406 LUTs, 402 FFs 1 BRAM 9 DSP48E	727 slices 15 BRAMs 16 DSP48
Max. clock frequency (IP)	202MHz	264MHz	-	-	125MHz	201MHz
Notes	Col/row filter swapped via DPR	Implemented with Altera DSP Builder	Reported: 36.78dB (Lena)			Xilinx ref. design

NC: number of columns, NR: number of rows.

Fig. 8. Different *lena* image sizes: VGA, CIF, and QCIF.

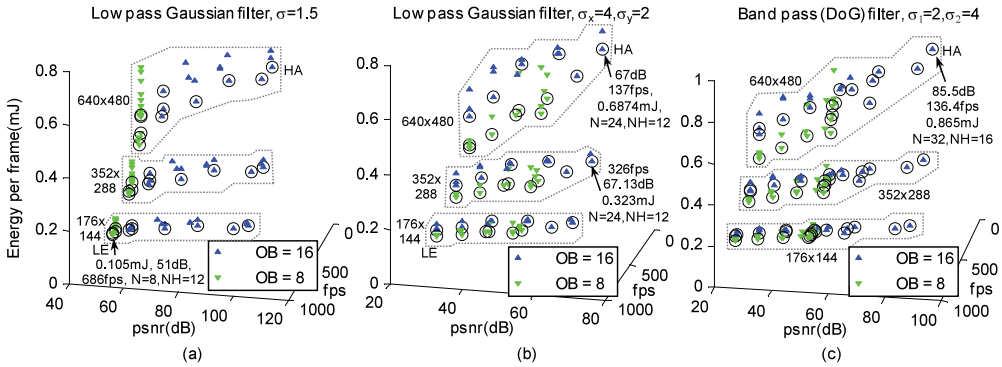

Fig. 9. EPA spaces and Pareto-optimal realizations (circled points) for three different *lena* image sizes and for three different filters: (a) Isotropic low-pass filtering. EPA space (90 realizations). (b) Anisotropic low-pass filtering. EPA space (90 realizations). (c) DOG filtering. EPA space (108 realizations). LE: lowest-energy realization. HA: highest-accuracy realization.

implementation does compare favorably against Yang et al. [2010], Bhandari et al. [2009], and Raikovich and Feher [2010].

6.2. Multiobjective Optimization of the EPA Space

In this section, we present our results for EPA optimization for all filter types. The image lena at VGA, CIF, and QCIF resolutions (see Figure 8) is used to test different frame sizes. For the DOG filter, refer to Table II for the different parameter combinations that are considered. For the considered spreads of the low-pass filters, the quantized coefficients gave zero (or near-zero) values for N $>$ 24 (note that $12 = 3\sigma_{max}$, $\sigma_{max} = 4$), and thus we will omit the case N $=$ 32 for these filters.

In what follows, we designate Pareto-optimal 2D filter realizations using "E," "P," and "A" for energy, performance, and accuracy values. Similarly, we use "L" for the lowest-possible value and "H" for the highest. Thus, "HA" refers to a realization with the highest accuracy. The filter realization with the highest performance is denoted by "HP." The filter realization with the lowest energy realization is given by "LE."

Figure 9 presents the results from EPA space optimization for all filter types and for each lena image resolution. Clearly, when using only 8 output bits (OB $=$ 8), we obtain our lowest-energy realizations and lowest-accuracy results (LE, LA). The highest accuracy is achieved by increasing the number of coefficients and the coefficient bitwidth, and with 16 output bits. A frame size increase causes an increase in energy per frame and a decrease in performance. Since the EPA space and Pareto front are very similar for all frame sizes, we next focus on one frame resolution (CIF).

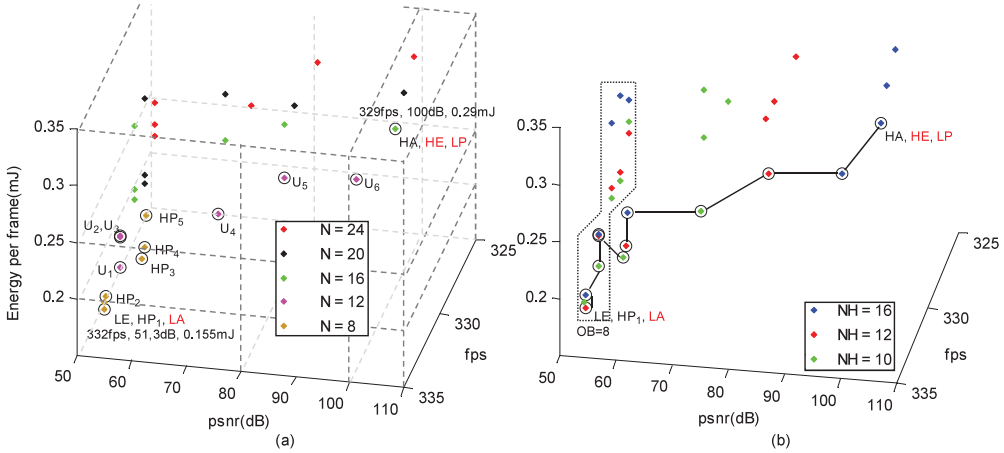

Fig. 10. Pareto front (12 realizations, circled points) for the isotropic low-pass Gaussian filter ($\sigma = 1.5$) for CIF resolution: (a) Graph showing dependence on the number of coefficients N. (b) Graph showing dependence on the coefficient bitwidth NH. Pareto-optimal points are circled. OB refers to the output bitwidth. Refer to text for definitions of HP, LE, HA, LP, HE, and LA.

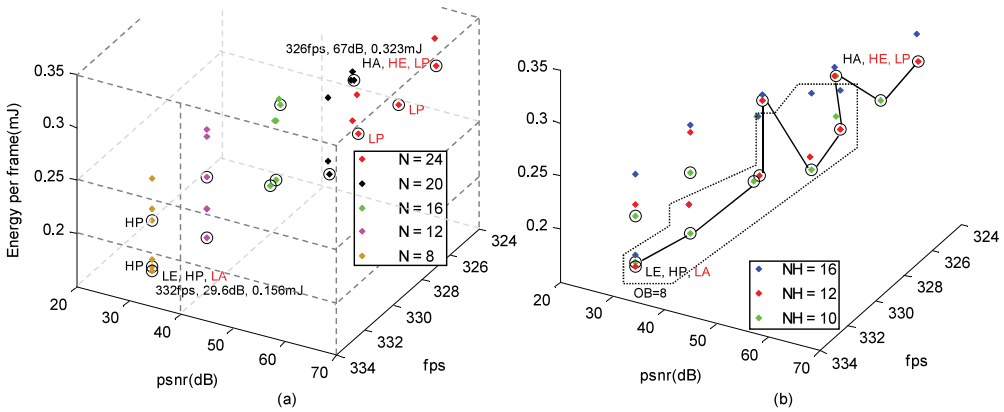

Fig. 11. Pareto front (13 realizations, circled points) for the anisotropic low-pass Gaussian filter ($\sigma_x = 4$, $\sigma_y = 2$) for CIF resolution: (a) Graph showing dependence on the number of coefficients N. (b) Graph showing dependence on the coefficient bitwidth NH. OB refers to the output bitwidth. Refer to text for definitions of HP, LE, HA, LP, HE, and LA.

Figures 10 through 12 show the EPA space optimization results for all filter types and for lena at CIF resolution. Each figure shows the Pareto-optimal realizations as a function of N, NH, and OB (OB = 8 is grouped in a polygon). In each case, we do not see much variation in performance. Performance variations only occur for different frame sizes as shown in Figure 9. Also, note that energy consumption increases with accuracy.

Even at the largest frame size (VGA), performance values are always over 100fps. For CIF resolution, performance exceeds 300fps. Overall, for a fixed frame size, there are slight variations in performance. Thus, in what follows, we will restrict our attention to the energy-accuracy space.

Figure 13 shows the results from energy-accuracy space optimization for all filter types and for *lena* at CIF resolution. Table V lists the 2D Pareto-optimal realizations

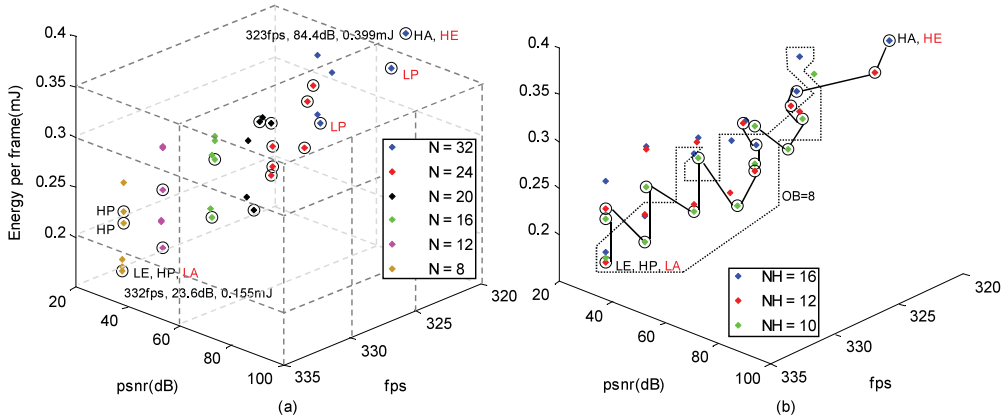

Fig. 12. Pareto front (19 realizations, circled points) for DOG filter ($\sigma_1 = 2$, $\sigma_2 = 4$) for CIF resolution: (a) Graph showing dependence on the number of coefficients N. (b) Graph showing dependence on coefficient bitwidth NH. OB refers to the number of output bits. Refer to text for definitions of HP, LE, HA, LP, HE, and LA.

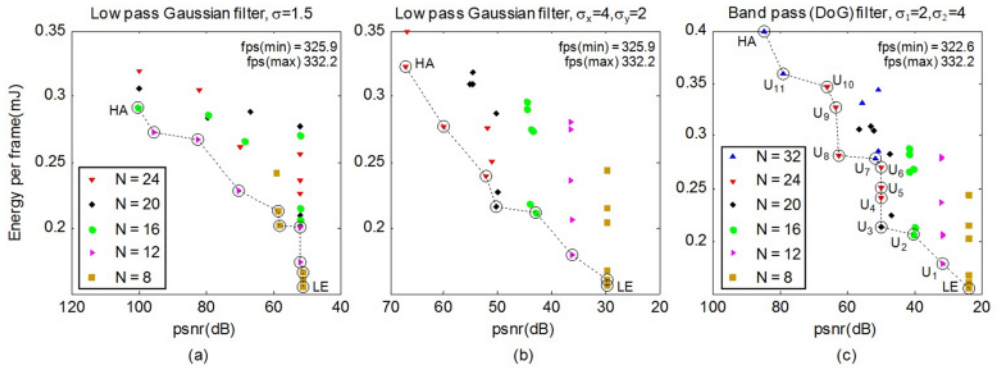

Fig. 13. 2D Pareto-optimal realizations (circled points) for Energy-Accuracy space for all filter types at CIF resolution: (a) Isotropic, low-pass filter. (b) Anisotropic, low-pass filter. (c) DOG filter. In all cases, we also summarize performance in terms of the minimum and maximum fps. Here, 'LE' refers to the lowest energy realization, and 'HA' refers to the highest accuracy realization.

Table V. Pareto-Optimal Realizations for DOG Filtering ($\sigma_1 = 2$, $\sigma_2 = 4$) of *lena* at CIF Resolution. Realizations are sorted by energy consumption. Refer to text for acronyms. U1–11 represent intermediate points.

	N	NH	OB	psnr (dB)	Energy per Frame (mJ)
LE	8	12	8	23.6345	0.1551
U1	12	10	8	31.7124	0.1780
U2	16	10	8	40.2984	0.2062
U3	20	10	8	49.5758	0.2139
U4	24	12	8	49.6447	0.2416
U5	24	10	8	49.8104	0.2508
U6	24	16	8	49.9967	0.2704
U7	32	10	8	51.6392	0.2781
U8	24	10	16	62.3297	0.2807
U9	24	12	16	63.1992	0.3276
U10	24	16	16	65.8199	0.3464
U11	32	12	16	78.7569	0.3586
HA	32	16	16	84.4371	0.3991

Table VI. Memory Requirements for Dynamic EPA Management

Filter Type	# of Pareto Points (# of bitstreams)	Bitstream Size	Required Memory
Isotropic, low-pass filter ($\sigma = 1.5$): Figure 13(a)	11 (22)	183KB	4.042MB
Anisotropic, low-pass filter ($\sigma_x = 4$, $\sigma_y = 2$): Figure 13(b)	8 (16)	183KB	2.940MB
DOG filter ($\sigma_1 = 2$, $\sigma_2 = 4$): Figure 13(c)	13 (26)	183KB	4.777MB

for the DOG filter. Note that accuracy increases with energy consumption and the number of output bits.

We next perform dynamic management by selecting realizations among the Pareto-optimal front. Table VI lists the memory requirements for the bitstreams of the Pareto front for each of the cases listed in Figure 13. For the cases of Figure 13(a–b), the largest filter realization has N = 24, NH = 16, and OB = 16. For the case of Figure 13(c), the largest filter realization has N = 32, NH = 16, and OB = 16. The resource consumption of the column filters with N = 24 (L = 6) and N = 32 (L = 4) is almost the same, as shown in Figure 7. Thus, for all three cases, we use the PRR that accommodates the column filter with N = 32, NH = 16, and OB = 16. From Section 6.1.2, the bitstream size for that reconfigurable area is 183,754 bytes. All cases listed in Table VI fit in the 128MB DDR-SDRAM inside the ML405 board.

6.3. Dynamic Reconfiguration Manager That Allows for Dynamic EPA Management

The embedded system of Figure 4(a) allows us to implement a dynamic manager (software routine in C running in the PowerPC processor) that can trigger automatic constraints based on the user input, filter output, or input image type (see Figure 4(c)). We can then demonstrate dynamic EPA management on two video sequences.

We present the case of dynamic reconfiguration based on the user input in Figures 14(a) and 14(b). We consider a time-varying sequence of energy-accuracy constraints for the DOG filter as listed on the top of Figure 14(a):

(1) Require accuracy $\geq$45dB and energy $\leq$0.3mJ per frame.
(2) Maximize accuracy subject to energy $\leq$0.3mJ per frame.
(3) Minimize energy consumption per frame.
(4) Minimize energy per frame subject to accuracy $\geq$65dB.
(5) Maximize accuracy.

The goal of our dynamic management approach is to meet the constraints by using Pareto-optimal realizations listed in Table V. Here, our assumption is that the accuracy results from the *lena* image will be close to those of the testing video sequence. As we see in Figures 14(a) and 14(b), this assumption seems to apply here. Nevertheless, even if the assumption does not hold, a user can dynamically adjust the constraints to match expectations.

Recall that when more than one realization meets the constraints, we pick the one with minimum energy (e.g., see point ①). The EPA constraints yielded the 2D FIR filter realizations shown in Figure 14(a). Figure 14(b) shows the dynamic manager sequentially applying the constraints every 60 frames for the foreman video sequence.

We present the case of dynamic reconfiguration based on the output frames in Figures 14(c) and 14(d). The state diagram for generating the real-time constraints is shown in Figure 14(c). The success of the approach is demonstrated in Figure 14(d). It is clear from Figure 14(d) that the scene change was detected and triggered a state

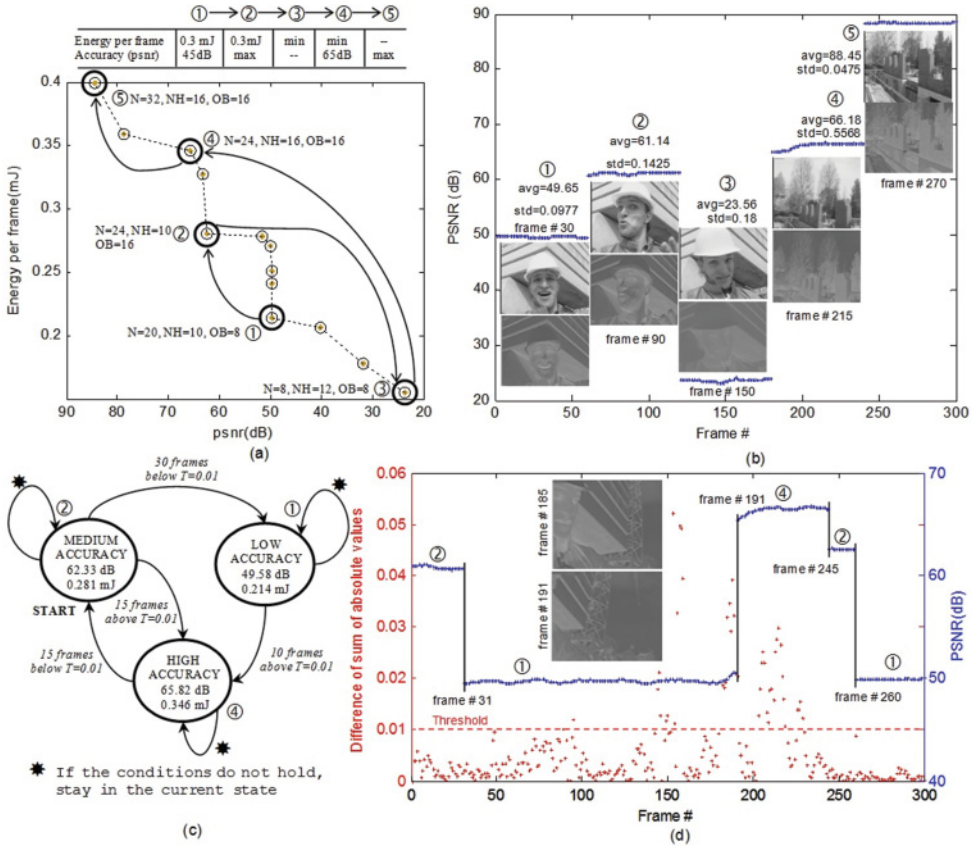

Fig. 14. Dynamic EPA management example for DOG filtering. Video sequence: *foreman* (CIF): (a) Sequence of 2D FIR realizations that meet user-defined dynamic constraints. The dynamic constraints are displayed above the graph. (b) Video filtering accuracy achieved by dynamic EPA management for meeting the constraints given in (a). (c) State diagram implementation for dynamic manager based on a percentage change of $T = 1\%$ (see Figure 6). The numbers on the states correspond to the hardware realizations of (a). (d) Detection of a scene change by using $T = 1\%$ (see frames 185 and 191).

transition from low- to high-accuracy mode using a threshold of 0.01. From Figure 14(d), we can also estimate that if the threshold were larger, fewer transitions would occur. Conversely, with a smaller threshold, more transitions are likely to occur.

Figure 15 depicts the case of dynamic reconfiguration based on input image type. Here, the I, P, and B frame types are associated with high-, medium-, and low-accuracy modes as demonstrated in Figure 15(c). The results were computed for the "news" video sequence (CIF).

6.4. A 2D Separable Filter Implementation Using Two PRRs

It is possible to implement a 2D separable filter where the row and column filter will always be present. We then define two PRRs (for the row and column filters). If the dynamic reconfiguration manager requires a different 2D filter, we reprogram these two PRRs. This approach requires far fewer reconfigurations and less memory to store the bitstreams (the PRR of the row filter will be smaller). However, the drawback of using a system with two PRRs is that it requires more resources and far more energy

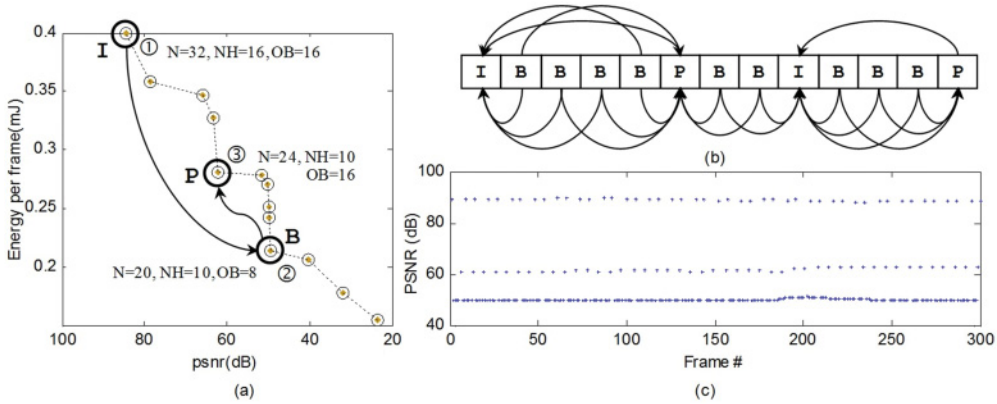

Fig. 15. Dynamic example on 'news' video sequence (CIF). (a) Pareto optimal front and the 3 points that correspond to I,P,B frames. (b) MPEG frame type ordering (and dependencies) that is used for determining the Pareto realizations. (c) Achieved PSNR for each video frame.

than our proposed system with one PRR. We illustrate this by using the power and resource results from our targeted FPGA (XC4VFX20).

The largest column filter requires 25% of FPGA resources, while the largest row filter requires about 13%. So, we require roughly 40% of the FPGA resources (taking into account any extra ancillary logic). The system for the largest 2D filter will need about 98% of the FPGA resources (58% for the static region). This means that we cannot have 2D filters with more than N = 32 coefficients if we were to use the system with two PRRs.

As for power consumption, the extra power required by the system with two PRRs is about 96mW (required by the idle PRR), while the extra power (reconfiguration) required by our system with one PRR is about 63mW. Note that our system with one PRR only spends the extra reconfiguration power when switching 1D filters, while the system with two PRRs will require the extra power all the time.

Thus, the proposed system with one PRR spends far less energy than the system with two PRRs. If, in the system with two PRRs, we were to shut down the idle PRR via DPR, there will be no advantage as we will need two reconfigurations per frame (as our system with one PRR).

6.5. A Comparison of the Dynamic FIR Filter Processor with Static Implementations

In this section, we provide a comparison of the dynamic FIR implementation to two static implementations. To make the comparison fair, we consider an efficient static implementation using DSP blocks and another implementation based on FPGA LUTs. The embedded system in the static implementation is based on a multiply-and-add 1D FIR core that implements the row and column filters by loading the coefficients into the DSP blocks (instead of performing DPR). The 1D FIR core resembles the hardware of Figure 3, but the outputs from the register chain go directly into a vector multiplier. Using the same definitions found in Section 3.1, the I/O latency is given by $REG\_LEVELS = \lceil \log_2 M \rceil + 2$. Prior to filtering, we need to feed the M coefficients into a register chain. Similar to the dynamic FIR implementation, we have row filtering, transposition, and column filtering.

We provide an extensive comparison between the dynamic and static implementations in Table VII for N = 32, NH = 16, and OB = 16. From the results, we can see that the dynamic filter system implementation is more accurate (about 3dB) than the static ones. This is due to the use of Distributed Arithmetic, which allows us to quantize

Table VII. EPA Comparison for a Symmetric DOG Filter with $N = 32$, $NH = 16$, and $OB = 16$. Device: XC4VFX20 Virtex-4 FPGA. In this system, we assume two reconfigurations per video frame.

	Accuracy (dB)			**Resources** (column filter)				**Power** (mW)	
	VGA	CIF	QCIF	FFs	LUTs	Slices	DSP48	Row	Column
Dynamic	85.48	84.43	84.49	3,680	3,511	2,076	0	99.62	120.70
Static (DSP48)	82.51	81.43	81.07	1,807	1,583	1,081	16	109.59	121.35
Static (no DSP)				1,807	6,847	3,529	0	156.81	173.82

	Energy per Frame (mJ)			**Performance**: t_{rows} (ms), t_{cols} (ms), fps					
	Dynamic	Static (with DSP)	Static (no DSP)	Dynamic			Static (with/without DSP)		
				t_{rows}	t_{cols}	fps	t_{rows}	t_{cols}	fps
VGA	0.8655	0.7655	1.0959	3.1872	3.2256	136.39	3.2784	3.3472	150.93
CIF	0.3991	0.2661	0.3809	1.0829	1.0982	322.59	1.1376	1.1651	434.27
QCIF	0.2230	0.0745	0.1067	0.2880	0.2956	665.58	0.3154	0.3291	1551.6

Table VIII. Energy per Frame (epf) and Performance (fps) for Various Frame Sizes and Various Values of K. 4CIF: 704×576, 720p: 1,280×720, 1,080p: 1,920×1,080, 2,000p: 2,048×1,536, 4,000p: 4,096×3,072. The static system uses DSP48s.

	STATIC SYSTEM		DYNAMIC SYSTEM							
Frame	epf		Energy per Frame (mJ)				Frames per Second			
Size	(mJ)	fps	K = 1	K = 5	K = 10	K = 20	K = 1	K = 5	K = 10	K = 20
QCIF	0.0745	1551.6	0.2230	0.0961	0.0802	0.0723	665.58	1303	1480	1588
CIF	0.2661	434.27	0.3991	0.2722	0.2563	0.2484	322.59	422.85	439.94	449.02
VGA	0.7655	150.93	0.8655	0.7386	0.7227	0.7148	136.39	151.59	153.73	152.83
4CIF	1.0004	115.467	1.0862	0.9593	0.9434	0.9355	107.11	116.27	117.52	118.16
720p	2.2291	51.8350	2.2434	2.1165	2.1006	2.0927	50.42	52.37	52.62	52.75
1,080p	4.9399	23.3852	4.8086	4.6817	4.6658	4.6579	23.19	23.60	23.65	23.67
2,000p	7.4441	15.5145	7.1853	7.0584	7.0425	7.0346	15.45	15.63	15.65	15.67
4,000p	29.4180	3.9256	28.073	27.95	27.930	27.923	3.932	3.944	3.945	3.946

the LUT values rather than the coefficients) [Llamocca et al. 2010]. The row filter is smaller than the column filter, but in the static systems, the row filter underutilizes the column filter resources. In Table VII, we show the resource consumption of the column filter for the dynamic and static (with and without DSP48s) systems. The static system based on DSP48 blocks requires fewer slices, though it uses 16 DSP48 blocks (50% of the available ones). The static system based on FPGA slices uses far more resources than our dynamic system. This is because the Distributed Arithmetic implementation is more efficient than a direct multiply-and-add based on FPGA slices. The video frame size has a negligible effect on the hardware resources.

In the static systems, since the hardware implementation is always that of the (largest) column filter, the power consumption of the row and column filters is about the same (the row filter spends slightly less power since some input ports are driven to zero). In the dynamic case, the power consumption of the column filter is marginally better than that of the static one (with DSP48). The hardwired DSP48s exhibit lower power consumption than the logic in the FPGA fabric. Notice that the DSP48 blocks are limited: if the filter were nonsymmetric with N = 32, we would require all of the 32 available DSP48s. So, our dynamic system is saving precious DSP48s that can be used elsewhere.

Performance and energy formulas (Equations (4) and (7)) apply for the static system (omitting the reconfiguration time). Performance-wise, we show t_{rows}, t_{cols}, and fps for all cases and for all frame sizes. In the static system, computing trows and tcols requires an additional M cycles in the inner summation of Equation (2), since we need to feed the M coefficients. These times are marginally better for the dynamic system. Since the dynamic system incurs a reconfiguration overhead, the static system operates at higher frame rates. It is the same for energy consumption: the static system based on DSP48 blocks consumes less energy per frame than the dynamic system.

The energy and performance (fps) values of Table VII assume that the dynamic system always reconfigures twice per frame (K = 1). If, however, we reconfigure every K frames (K > 1; see discussion in Section 3.3) and/or increase the video frame size, we can reduce the reconfiguration time overhead and in turn improve the energy and performance values. Table VIII shows energy and performance values for various frame sizes and for K = 1, 5, 10, 20.

The dynamic system provides lower energy consumption for frame sizes 1,080p (K = 1), VGA (K = 5), and CIF (K = 10). As for performance (fps), the dynamic system is better than the static one (DSP48) for frame sizes 4,000p (K = 1), 1,080p (K = 5), and 4CIF (K = 10). When K = 20 or frame size is 4,000p, the dynamic system is better than the static system in both energy and performance (fps). This effectively demonstrates

Table IX. Total Energy (mJ) for the Applications Depicted in Figures 14(b), 14(d), and 15(c) as a Function of Frame Size. Figures 14(b), 14(d), and 15(c) show the case for CIF frame size. The dynamic cases of QCIF and VGA were computed in a similar fashion since we have the Pareto front (Figure 9). The other dynamic cases—4CIF, 720p, 1,080p, 2,000p, and 4,000p are conservative projections, assuming that the energy per frame just scales with frame size. $K = 1$.

	Static System (with DSP48s)	Dynamic System		
Frame Size	**Figure 14(b)/ Figure 14(d)/Figure 15(c)**	**Figure 14(b)**	**Figure 14(d)**	**Figure 15(c)**
QCIF	22.3501	49.4392	44.8891	45.6739
CIF	79.8186	83.7069	74.3222	75.9370
VGA	229.6431	174.8532	152.7153	156.4436
4CIF	300.1105	219.4482	191.6642	196.3434
720p	668.7316	453.2375	395.8538	405.5180
1,080p	1,480.20	971.4961	848.4964	869.2111
2,000p	2,233.20	1,451.70	1,267.90	1,298.80
4,000p	8,825.40	5,671.70	4,953.60	5,074.60

that a frame size increase and/or an increment in K offset the reconfiguration overhead to the point that the dynamic system becomes the best solution. In other words, the dynamic system becomes more efficient than the static one (using DSP48s) as the frame size increases and/or K increases.

In terms of the total energy consumption, we summarize the results in Table IX. Here, we repeat the experiments of Figures 14(b), 14(d), and 15(c) for different video frame sizes. From Figures 14(b), 14(d), and 15(c), the total energy spent for processing 300 CIF-size frames is 83.71, 74.32, and 75.94mJ, respectively. The static system (with DSP48s) would have spent 79.83mJ. For the cases of video frame sizes of 4CIF to 4,000p, we assume that the energy per frame scales linearly with the number of pixels of the VGA base case. Similarly, the Pareto points used for the different video frame sizes are assumed to be the ones generated from the CIF case described earlier. For QCIF and VGA video frame sizes, the total energy consumption is obtained from the values depicted in Figure 9. For the results, we have $K = 1$. Starting with the VGA case, we can see that the dynamic system provides energy savings of more than 30%.

The quantities in Tables VII and VIII assume that we are implementing only one 2D FIR filter. The greatest disadvantage of the static system is its inability to adapt to different 2D FIR filters. We can think of a few instances in which we can reuse the hardware of the largest column filter to implement another filter with different parameters (N, NH, OB). However, for most cases, this is not possible (e.g., switching from a symmetric filter with $N = 31$ to a one with $N = 32$ would require the inclusion of an extra adder). Even for the few instances in which this would be possible, we will always consume more or less the energy of the (largest) column filter hardware (unlike the dynamic system that can effectively adapt energy consumption).

For small video frame sizes, the static 2D FIR filter system based on DSP48 blocks has some advantages over the dynamic system. In terms of total energy consumption, for modern video systems that require processing of larger video frames, the proposed approach performs significantly better (more than 30% for VGA sizes or larger). Also, the reconfiguration overhead can be controlled by restricting dynamic reconfiguration every K frames ($K > 1$). Overall, for larger video frame sizes and lower reconfiguration rates, significant reductions in energy consumption are possible with the proposed dynamic system by switching to lower-energy configurations.

7. CONCLUSIONS

We have presented a 2D FIR filtering framework for determining Pareto-optimal realizations in the energy-performance-accuracy (EPA) space. We also demonstrated how

Pareto-optimal realizations can be used to meet time-varying EPA constraints. This is an effective method for dynamic EPA management of video processing hardware.

We presented results over three 2D Gaussian filters. In each case, we provide a collection of Pareto-optimal solutions based on maximizing accuracy and performance while minimizing consumption of energy per frame.

Dynamic EPA management is demonstrated on two standard video sequences. A dynamic reconfiguration manager can generate time-varying constraints based on the input or the output video frames. Here, it was clearly demonstrated how energy-accuracy constraints can be easily met using a precomputed set of Pareto-optimal realizations.

More complex applications are likely to benefit significantly from the proposed approach because image processing time will be significantly larger than the reconfiguration time. In other words, it is far more beneficial to reconfigure to an optimized architecture that will be used over a much longer time in a complex application, as opposed to the simpler applications considered in the current article. The use of more parameters in more complex applications will likely grow the amount of offline computation needed to generate the Pareto front. In this case, a partial search approach can be used to reduce the size of the Pareto front. In the real-time implementation, for a very large Pareto front, it makes sense to consider the implementation of a cache system for effective retrieval of the Pareto-optimal configurations.

The results suggest that this framework can be applied to a variety of video processing systems. The implementation of the dynamic manager heavily depends on the application and is a very interesting topic for further research in this area. Overall, the dynamic system is very effective for usage with larger video frame sizes and lower reconfiguration rates.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under NSF AWD CNS-1422031.

REFERENCES

Afandi Ahmad and Abbes Amira. 2009. Efficient reconfigurable architectures for 3D medical image compression. In *Proceedings of International Conference on Field-Programmable Technology.* 472–474.

M. S. Andrews. 1999. Architectures for generalized 2d FIR filtering using separable filter structures. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '99).* 2215–2218.

Juergen Becker, Michael Huebner, and Michael Ullmann. 2003. Power estimation and power measurement of Xilinx Virtex FPGAs: Tradeoffs and limitations. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*. 283–288.

Sheetal U. Bhandari, Shaila Subbaraman, Shashank S. Pujari, and Rashmi Mahajan. 2009. Real time video processing on FPGA using on the fly partial reconfiguration. In *Proceedings of the International Conference on Signal Processing Systems.* 244–247.

Robin Bonamy, Daniel Chillet, Sébastien Bilavarn, and Olivier Sentieys. 2012. Power consumption model for partial and dynamic reconfiguration. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig'12).* 1–8.

Kiran Bondalapati and Viktor K. Prasanna. 1999. Dynamic precision management for loop computations on reconfigurable architectures. In *Proceedings of the 7th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.* 249–258.

Sophie Bouchoux, El-Bay Bourennane, and Michael Paindavoine. 2004. Implementation of JPEG2000 arithmetic decoder using dynamic reconfiguration of FPGA. In *Proceedings of the International Conference on Image Processing.* 2841–2844.

Christos-S. Bouganis, Sung-Boem Park, George A. Constantinides, and Peter Y.K. Cheung. 2009. Synthesis and optimization of 2D filter designs for heterogeneous FPGAs. *ACM Trans. Reconfig. Technol. Syst*. 1, 4, Article 24 (Jan. 2009), 28 pages. DOI:http://dx.doi.org/10.1145/1462586.1462593

Alan Bovik (Ed.). 2009a. *The Essential Guide to Image Processing* (2nd ed.). Academic Press, Elsevier.

Alan Bovik (Ed.). 2009b. *The Essential Guide to Video Processing* (2nd ed.). Academic Press, Elsevier.

Wayne Burleson, Prashant Jain, and Subramanian Venkatraman. 2001. Dynamically parameterized architecture for power-aware video coding: Motion Estimation and DCT. In *Proceedings of 2nd USF International Workshop on Digital and Computational Video*. 8–12.

Roger Chamberlain, Eric Hemmeter, Robert Morley, and Jason White. 2003. Modeling the power consumption of audio signal processing computations using customized numerical representations. In *Proceedings of the 36th Annual Simulation Symposium*. 249–255.

C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker. 2008. A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 535–538.

Omkar Dandekar, William Plishker, Shuvra S. Bhattacharyya, and Raj Shekkar. 2008. Multiobjective optimization for reconfigurable implementation of medical image registration. *Int. J. Reconfig. Comput.* 2008, Article ID 738174, 17 pages.

Mariano Fons, Francisco Fons, and Enrique Cantó. 2010. Fingerprint image processing acceleration through run-time reconfigurable hardware. *IEEE Trans. Circuits Syst. II: Express Brief*, 57, 12 (Dec. 2010), 991–995. DOI:http://dx.doi.org/10.1109/TCSII.2010.2087970

Hugo Hedberg, Petr Dokladal, and Viktor Öwall. 2009. Binary morphology with spatially variant structuring elements: Algorithm and architecture. *IEEE Trans. Image Process.* 18, 3 (March 2009), 562–572. DOI:http://dx.doi.org/10.1109/TIP.2008.2010108

John C. Hoffman and M. Marios Pattichis. 2011. A high-speed dynamic partial reconfiguration controller using direct memory access through a multiport memory controller and overclocking with active feedback. *Int. J. Reconfig. Comput.* 2011, Article ID 439072, 10 pages.

Sangjin Hong, Jinseok Lee, Akshay Athalye, Petar M. Djurić, and We-Duke Cho, W. 2007. Design methodology for domain specific parameterizable particle filter realizations. *IEEE Trans. Circuits Syst. Part I: Regular Papers*, 54, 9 (Sep. 2007), 1987–2000. DOI:http://dx.doi.org/10.1109/TCSI.2007.904690

Yohei Hori, Akashi Satoh, Hirofumi Sakane, and Kenji Toda. 2008. Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 23–28.

Jian Huang and Jooheung Lee. 2009. A self-reconfigurable platform for scalable DCT computation using compressed partial bitstreams and BlockRAM prefetching. *IEEE Trans. Circuits Syst. Video Technol.* 19, 11 (Nov. 2009), 1623–1632. DOI:http://dx.doi.org/10.1109/TCSVT.2009.2031464

Takashi Komuro, Tomohira Tabata, and Masatoshi Ishikawa. 2010. A reconfigurable embedded system for 1000 f/s real-time vision. *IEEE Trans. Circuits Syst. Video Technol.* 20, 4 (April 2010), 496–504. DOI:http://dx.doi.org/10.1109/TCSVT.2009.2035832

Jahyun Koo, Alan C. Evans, and Warren J. Gross. 2009. 3-D brain MRI tissue classification on FPGAs. *IEEE Trans. Image Process.* 18, 12 (Dec. 2009), 2735–2746. DOI:http://dx.doi.org/10.1109/TIP.2009.2028926

Ming Liu, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch. 2009. Run-time partial reconfiguration speed investigation and architectural design space exploration. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 498–502.

Yong Liu and Edmund M-K. Lai. 2004. Design and Implementation of an RNS-based 2-D DWT processor. *IEEE Trans. Consum. Electron.* 50, 1 (Feb. 2004), 376–385. DOI:http://dx.doi.org/10.1109/TCE.2004.1277887

Daniel Llamocca and Marios Pattichis. 2013. A dynamically reconfigurable pixel processor system based on power/energy-performance-accuracy optimization. *IEEE Trans. Circuits Syst. Video Technol*. 23, 3 (March 2013), 488–502. DOI:http://dx.doi.org/10.1109/TCSVT.2012.2210664

Daniel Llamocca, Marios Pattichis, and Alonzo Vera. 2010. Partial reconfigurable FIR filtering system using distributed arithmetic. *Int. J. Reconfig. Comput.* 2010, Article ID 357978, 14 pages.

Daniel Llamocca and Marios Pattichis. 2010. Real-time dynamically reconfigurable 2-D filterbanks. In *Proceedings of 2010 IEEE Southwest Symposium on Image Analysis & Interpretation*. 181–184.

Daniel Llamocca, Cesar Carranza, and Marios Pattichis. 2011. Separable FIR filtering in FPGA and GPU implementations: Energy, performance, and accuracy considerations. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 363–368.

Daniel Llamocca, Cesar Carranza, and Marios Pattichis. 2012. Dynamic multiobjective optimization management of the energy-performance-accuracy space for separable 2-D complex filters. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 579–582.

David G. Lowe. 1999. Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer Vision*. 1150–1157.

David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110.

Hong S. Neoh and Asher Hazanchuk. 2004. Adaptive edge detection for real-time video processing using FPGAs. In *Proceeding of the GSPx2004 Conference.*

Duy Nguyen, David Halupka, Parham Aarabi, and Ali Sheikholeslami. 2006. Real-time face detection and lip feature extraction using field programmable gate arrays. *IEEE Trans. Syst. Man Cybern.* 36, 4 (Aug. 2006), 902–912. DOI:http://dx.doi.org/10.1109/TSMCB.2005.862728

Juanjo Noguera and Irwin O. Kennedy. 2007. Power reduction in network equipment through adaptive partial reconfiguration. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 240–245.

Erdal Oruklu and Jafar Saniie. 2009. Dynamically reconfigurable architecture design for ultrasonic imaging. *IEEE Trans. Instrum. Meas.* 58, 8 (Aug. 2009), 2856–2866. DOI:http://dx.doi.org/10.1109/TIM.2009.2016370

Tamás Raikovich and Béla Fehér. 2010. Application of partial reconfiguration of FPGAs in image processing. In *Proceedings of 2010 Conference on Ph.D. Research in Microelectronics and Electronics.* 1–4.

Javier Resano, Daniel Mozos, Diederik Verkest, Serge Vernalde, and Francky Catthoor. 2003. Runtime minimization of reconfiguration overhead in dynamically reconfigurable systems. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 585–594.

Arvind Sudarsanam, Robert Barnes, Jeff Carver, Ramachandra Kallam, and Aravind Dasu. 2010. Dynamically reconfigurable systolic array accelerators: A case study with extended Kalman filter and discrete wavelet transform algorithms. *IET Comput. Digit. Tech*. 4, 2 (March 2010), 126–142. DOI:http://dx.doi.org/10.1049/iet-cdt.2008.0139

Robert Turney. 2007. *Two-Dimensional Linear Filtering* (XAPP933). v1.1 ed., Xilinx Inc., San Jose, CA.

Alonzo Vera, Marios Pattichis, and James Lyke. 2011. A dynamic dual fixed-point arithmetic architecture for FPGAs. *Int. J. Reconfig. Comput.*, 2011, Article ID 518602, 19 pages.

Xilinx. 2010a. *Partial Reconfiguration User Guide for ISE 12.3* (UG702). v12.3 ed., Xilinx Inc., San Jose, CA.

Xilinx. 2010b. *Virtex-6 Family Overview* (DS150). v2.2 ed., Xilinx Inc., San Jose, CA.

Xilinx 2011. *Power Methodology Guide* (UG786). v13.1 ed., Xilinx Inc., San Jose, CA.

Huaqiu Yang, Fanjiong Zhang, JinMei Lai, and Yan Wang. 2010. Image filtering using partially and dynamically reconfiguration. In *Proceedings of the International Conference on Solid-State and Integrated Circuit Technology.* 2067–2073.

Received July 2012; revised June 2013, December 2013; accepted April 2014